UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/862,654	05/22/2001	Nigel Peter Topham	0808.65559	8723

24978 7590 09/10/2004

GREER, BURNS & CRAIN
300 S WACKER DR
25TH FLOOR
CHICAGO, IL 60606

EXAMINER	
O#BRIEN, BARRY J	

ART UNIT	PAPER NUMBER
2183	

DATE MAILED: 09/10/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/862,654	TOPHAM, NIGEL PETER	
	Examiner	Art Unit	
	Barry J. O'Brien	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *29 June 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-13 and 15-51* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-13 and 15-51* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☒ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☒ Other: *See Continuation Sheet*.

Continuation of Attachment(s) 6). Other: Office Flyer (Revised Amendment Practice).

DETAILED ACTION

1. Claims 1-13 and 15-51 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed on record in the file: Amendment A as received on 6/29/2004.

Specification

3. The lengthy specification has not been checked to the extent necessary to determine the presence of all possible minor errors. Applicant's cooperation is requested in correcting any errors of which applicant may become aware in the specification.

4. The applicant is requested to review the specification and update the status of all co-pending applications made mention of, replacing attorney docket numbers with current U.S. application or patent numbers when appropriate.

Improper Amendment

5. Applicant's amendment of claim 27 in the present amendment, filed 6/29/04, has failed to adhere to 37 CFR 1.121. As is clearly seen from the attached office flyer, "The text of all claims being currently amended must be presented in the claim listing with markings to indicate the changes that have been made relative to the immediate prior version". Applicant's amendment fails to identify claim language amended in claim 27 correctly.

6. The last paragraph of the prior version of claim 27 read, "storing, in said program memory, the compressed-form instructions together with imaginary address information specifying said assigned imaginary addresses so that, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache, the processor can assign the specified imaginary addresses to the decompressed instructions." However, words that did not exist in the prior version of claim 27 are indicated in the amended version of claim 27 as being cancelled (see "comprising" on the second line of the last paragraph of amended claim 27), as well as words that did not exist in the prior version being indicated as having previously existed (see "storable" on the first line of the last paragraph of amended claim 27).

7. Applicant is advised to carefully review the requirements of 37 CFR 1.121 regarding amendment formatting as summarized in the attached flyer, and to assure that all future amendments properly adhere to these requirements. While not being held as an improper amendment, the Applicant is advised to correct this error on subsequent amendments.

Claim Objections

8. Claim 43 is objected to because of the following informalities:

   a. Claim 43 recites the limitation, "loaded by the processor into the instruction cache the processor can allocate the decompressed instructions" on its last three lines. It is unclear what "the instruction cache the processor" means, as it is not grammatically correct English. Please correct the claim language to fix this apparent grammatical error.

Appropriate correction is required.

Claim Rejections - 35 USC § 102

9. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

10. Claim 1-2, 5, 10-13, 15-18, 20, 23 and 37 are rejected under 35 U.S.C. 102(b) as being anticipated by Faraboschi et al., U.S. Patent No. 5,870,576.

11. Regarding claim 1, Faraboschi has taught a processor, for executing instructions of a program stored in compressed form (see Col.4 lines 45-46, 58-67) in a program memory (see 110 of Fig.3), comprising:

a. A program counter (132 of Fig.2), which identifies a position in said program memory (see Col.5 lines 1-15 and Col.6 lines 15-27),

b. An instruction cache (100 of Fig.3), having a plurality of cache blocks (see Fig.2), each for storing one or more instructions of said program in decompressed form (see Col.4 lines 5-8 and Col.5 lines 29-35),

c. A cache loading unit (204/210/212 of Fig.3), comprising a decompression section (210/212 of Fig.3), operable to perform a cache loading operation in which one or more compressed-form instructions are read from said position in the program memory identified by the program counter and are decompressed and stored in one of said cache blocks of the instruction cache (see Col.5 lines 24-34 and Col.6 lines 11-53), the compressed-form

instructions being stored in the program memory in one or more compressed sections (see Fig.2), and the compressed-form instructions belonging to each section occupying one of said cache blocks when decompressed (see Col.4 lines 48-67 and Col.5 lines 29-32). Here, the program memory contains compressed-form instructions (W00-W56 of Fig.2) stored in a "compressed section" of the heap (see address 14000300 in heap of Fig.2), which are uncompressed and stored in a cache block of the cache (see WOO in row 040 of Fig.2).

d. A cache pointer (200 of Fig.3), which identifies a position in said instruction cache of an instruction to be fetched for execution (see Col.5 lines 19-21, 55-57),

e. An instruction fetching unit (208/220 of Fig.3) which fetches an instruction to be executed from the position identified by the cache pointer (see Col.5 lines 36-40) and which, when a cache miss occurs because the instruction to be fetched is not present in the instruction cache, causes the cache loading unit to perform said cache loading operation (see Col.5 lines 40-50),

f. An updating unit (206 of Fig.3) which updates the program counter (132 of Fig.2) and cache pointer (200 of Fig.3) in response to the fetching of instructions so as to ensure that said position identified by said program counter is maintained consistently at the position in said program memory at which the instruction to be fetched form the instruction cache is stored in compressed form (see Col.5 lines 19-24 and Col.6 lines 11-53), said

updating unit comprising a next-section locating section operable, in the event of such a cache miss, to employ next-section-locating information (150, 152 and 200 of Fig.2), stored within the processor (see Figs.2, 3) in association with the cache block which was accessed most recently to fetch an instruction (see Fig.2 and Col.5 lines 48-50, where address in program counter is stored in the cache tag corresponding to a cache block), to locate the position in the program memory of a next compressed section following the compressed section corresponding to that most-recently-accessed cache block (see Col.6 lines 11-26). Here, the program counter (132 of Fig.2), which is comprised of a mask (150 of Fig.2) and an offset (152 of Fig.2), are updated to point to the current position in the program memory when performing an instruction fetch upon a cache miss (see Col.6 lines 11-53). Thus, along with the updating of the cache pointer (see Col.5 lines 19-24), the position of the current instruction to be fetched from the instruction cache is maintained. Further, on a cache miss, the instruction address in the program counter (200 of Fig.3) is used to access the code pointer segment (130 of Fig.2) in program memory (see Col.6 lines 11-16). The code pointer (152 of Fig.2) is then used to locate the position in the program memory of the next compressed section following the compressed section corresponding to the most-recently-accessed cache block (see Col.6 lines 16-24). The position located in the compressed memory is considered the next compressed section because the

compressed sections are arranged in consecutive memory locations (see Col.6 lines 20-24).

12. Regarding claim 2, Faraboschi has taught a processor as claimed in claim 1, wherein said position in the instruction cache of an instruction to be fetched is identified by said cache pointer (200 of Fig.3) in terms of an imaginary address assigned to the instruction (see Col.5 lines 19-21), at which the instruction is considered to exist when held in decompressed form in one of said cache blocks (see Fig.2, Col.4 lines 5-8 and Col.5 lines 29-32).

13. Regarding claim 5, Faraboschi has taught a processor as claimed in claim 1, wherein:

a. At least one section also contains imaginary address information relating to the instructions belonging to the section (see Fig.2). Here, the imaginary address information is stored in the code pointer (152 of Fig.2).

b. Said cache loading unit is operable, in said cache loading operation, to decompress and load into one of said cache blocks one such compressed section stored at the position in the program memory identified by the program counter (see Col.6 lines 19-35). Here, the cache refill state machine (204 of Fig.3), expansion logic (210 of Fig.3) and refill buffer (212 of Fig.3) perform a cache loading operation.

14. Regarding claim 10, Faraboschi has taught a processor as claimed in claim 5, wherein each said compressed section contains imaginary address information (152 of Fig.2) relating to the instructions belonging to the section concerned. Here, the code

pointer (152 of Fig.2) contains imaginary address information relating to the compressed instructions belonging to the section concerned.

15. Regarding claim 11, Faraboschi has taught a processor as claimed in claim 5, wherein the or each said compressed section further contains a decompression key (150 of Fig.2) which is employed by said decompression section (210 of Fig.2) to effect the decompression of the instructions belonging to the compressed section during the cache loading operation (see Col.6 lines 29-34).

16. Regarding claim 12, Faraboschi has taught a processor as claimed in claim 11, wherein the instructions of said program include, prior to compression, preselected instruction (NOP instructions indicated by the empty locations in cache of Fig.2) that are not stored explicitly in any said compressed section (see 140 of Fig.2), and the decompression key (150 of Fig.2) of the or each compressed section identifies the positions at which the preselected instructions are to appear in the cache block when the compressed section is decompressed (see Col.5 lines 4-13). Here, the NOP instructions are not stored in the main (compressed) memory.

17. Regarding claim 13, Faraboschi has taught a processor as claimed in claim 12, wherein said preselected instructions are "no operation" instructions (see Col.5 lines 4-13).

18. Regarding claim 15, Faraboschi has taught a processor as claimed in claim 1, wherein such next-section-locating information is stored in association with each cache block in which valid decompressed instructions are held (see Fig.2), the stored next-section-locating information being for use in locating the position in the program memory of said next compressed section (see Col.6 lines 11-24). Here, section-loading

information is stored in the cache-tag, mask and code pointer associated with the cache block in which there are valid instructions (see Fig.2). On a cache miss, the instruction address in the program counter is used to access the code pointer segment (130 of Fig.2) in the memory (see Col.6 lines 11-16). The code pointer is then used to locate the position in the memory of the next compressed section following the compressed section corresponding to that most-recently-accessed cache block (see Col.6 lines 16-24). The position located in the compressed memory is the next compressed section because compressed sections are arranged in consecutive memory locations (see Col.6 lines 202-24).

19. Regarding claim 16, Faraboschi has taught a processor as claimed in claim 15, wherein the next-section-locating information is stored in association with each cache block when that block is loaded in such a cache loading operation (see Col.6 lines 11-53). Here, although not taught explicitly, the next-section locating information is inherently stored in association with each cache block. The information in the cache tag (120 of Fig.2) must contain the address of the cache block with which it is associated so that it can be determined whether there was a cache hit or miss while comparing it with the instruction address (see Col.5 lines 48-50). So once the cache is loaded with the new block on a cache miss, the new address information must also be stored in the cache tag for proper working of the system.

20. Regarding claim 17, Faraboschi has taught a processor as claimed in claim 1, wherein said next-section-locating information associated with the cache block relates to a size of the compressed section corresponding to that cache block (see Col.6 lines 37-39). Here, the mask (150 of Fig.2) indicates the number of operations in the wide

instruction word, i.e. the size of the compressed section, which is to be stored in the cache block.

21. Regarding claim 18, Faraboschi has taught a processor as claimed in claim 17, wherein said size is determined by the cache loading unit when loading the cache block in the cache loading operation (see Col.6 lines 27-39). Here, the size is determined during the cache loading operation by the cache refill unit because it uses the mask to find out how many instructions are to be loaded into the instruction cache.

22. Regarding claim 20, Faraboschi has taught a processor as claimed in claim 1, wherein:

a. Said next-section-locating information associated with the cache block represents the number of instructions held in that cache block (see Col.5 lines 5-7 and Col.6 lines 37-39) that are not said preselected instructions. Here, the NOP instructions are not stored in the compressed section (140 of Fig.2) of the memory (110 of Fig.2).

23. Regarding claim 23, Faraboschi has taught a processor as claimed in claim 1, wherein the instructions of said program comprise very-long-instruction-word (VLIW) instructions (see Col.4 lines 5-21).

24. Regarding claim 37, Faraboschi has taught a processor, for executing instructions of a program stored in compressed form in a program memory (110 of Fig.3, see also Col.4 lines 45-46), comprising:

a. A program counter (132 of Fig.2) for identifying a position in said program memory (see Col.5 lines 1-15 and Col.6 lines 15-27),

b. An instruction cache (100 of Fig.3), having a plurality of cache blocks (see Fig.2), each for storing one or more instructions of said program in decompressed form (see Col.4 lines 5-8 and Col.5 lines 29-35),

c. Cache loading means (204/210/212 of Fig.3), including decompression means (210/212 of Fig.3), operable to perform a cache loading operation in which one or more compressed-form instructions are read from said position in the program memory identified by the program counter and are decompressed and stored in one of said cache blocks of the instruction cache (see Col.5 lines 24-34 and Col.6 lines 11-53), the compressed-form instructions being stored in the program memory in one or more compressed sections (see Fig.2), and the compressed-form instructions belonging to each section occupying one of said cache blocks when decompressed (see Col.4 lines 48-67 and Col.5 lines 29-32). Here, the program memory contains compressed-form instructions (W00-W56 of Fig.2) stored in a "compressed section" of the heap (see address 14000300 in heap of Fig.2), which are uncompressed and stored in a cache block of the cache (see WOO in row 040 of Fig.2).

d. A cache pointer (200 of Fig.3) for identifying a position in said instruction cache of an instruction to be fetched for execution (see Col.5 lines 19-21, 55-57),

e. Instruction fetching means (208/220 of Fig.3) for fetching an instruction to be executed from the position identified by the cache pointer (see Col.5 lines 36-40) and operable, when a cache miss occurs because the

instruction to be fetched is not present in the instruction cache, to cause the cache loading means to perform such a cache loading operation (see Col.5 lines 40-50),

f. Updating means (206 of Fig.3) for updating the program counter (132 of Fig.2) and cache pointer (200 of Fig.3) in response to the fetching of instruction so as to ensure that said position identified by said program counter is maintained consistently at the position in said program memory at which the instruction to be fetched form the instruction cache is stored in compressed form (see Col.5 lines 19-24 and Col.6 lines 11-53), said updating means comprising next-section locating means operable, in the event of such a cache miss, to employ next-section-locating information (150, 152 and 200 of Fig.2), stored within the processor (see Figs.2, 3) in association with the cache block which was accessed most recently to fetch an instruction (see Fig.2 and Col.5 lines 48-50, where address in program counter is stored in the cache tag corresponding to a cache block), to locate the position in the program memory of a next compressed section following the compressed section corresponding to that most-recently-accessed cache block (see Col.6 lines 11-26). Here, the program counter (132 of Fig.2), which is comprised of a mask (150 of Fig.2) and an offset (152 of Fig.2), are updated to point to the current position in the program memory when performing an instruction fetch upon a cache miss (see Col.6 lines 11-53). Thus, along with the updating of the cache pointer (see Col.5 lines 19-24), the position of the current instruction to be fetched

from the instruction cache is maintained. Further, on a cache miss, the instruction address in the program counter (200 of Fig.3) is used to access the code pointer segment (130 of Fig.2) in program memory (see Col.6 lines 11-16). The code pointer (152 of Fig.2) is then used to locate the position in the program memory of the next compressed section following the compressed section corresponding to the most-recently-accessed cache block (see Col.6 lines 16-24). The position located in the compressed memory is considered the next compressed section because the compressed sections are arranged in consecutive memory locations (see Col.6 lines 20-24).

Claim Rejections - 35 USC § 103

25. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

26. Claims 3-4, 6-9, 19, 21, 27-35 and 38-51 are rejected under 35 U.S.C. 103(a) as being unpatentable over Faraboschi et al., U.S. Patent No. 5,870,576.

27. Regarding claim 3, Faraboschi has taught a processor as claimed in claim 2, but has not explicitly taught wherein said imaginary address of an instruction is assigned thereto during assembly/linking of said program based on the sequence of original instructions in the program prior to compression.

28. However, "Official Notice" is taken that it is well known and expected in the art to use a linker to assign addresses to a program. This is substantiated by the definition of a liner provided by http://www.webopedia.com/TERM/l/linker.html (pg.2 lines 14-17). Therefore, one of ordinary skill in the art would have found it obvious that the imaginary addresses (successive instruction addresses, see Col.5 lines 19-21) must be assigned during the linking of the program based on the original sequence prior to compression, as it is well known that addresses are assigned during linking.

29. Regarding claim 4, Faraboschi has taught a processor as claimed in claim 2, has not explicitly taught wherein imaginary address information, from which said imaginary address assigned to each instruction is derivable, is stored with the compressed-form instructions in the program memory and is employed in the cache loading operation so as to associate with each decompressed instruction present in the instruction cache the imaginary address assigned thereto.

30. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see Fig.2 where code pointer segments have addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art would have recognized that by storing the address information along with the compressed instructions in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions from the memory. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address

information along with the compressed instruction, thus allowing for faster lookups and shorter clock periods using simpler circuitry.

31. Regarding claim 6, Faraboschi has taught a processor as claimed in claim 5, but has not explicitly taught wherein said imaginary address information of said at least one section specifies the imaginary address at which a first one of the decompressed instructions corresponding to the compressed section is considered to exist when the decompressed instructions are held in one of the cache blocks.

32. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see Fig.2 where code pointer segments have addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art would have recognized that by storing the address information along with the compressed instructions in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading instructions from the memory. This information would specify the imaginary address of the first one of the decompressed instructions because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the section in memory in order to provide for faster lookups and shorter clock periods using simpler circuitry.

33. Regarding claim 7, Faraboschi has taught a processor as claimed in claim 5, but has not explicitly taught wherein in said cache loading operation the cache block into

which the decompressed instructions of the compressed section are loaded is assigned an imaginary block address based on said imaginary address assigned to an instruction contained in the section being loaded.

34. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see Fig.2 where code pointer segments have addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art would have recognized that by storing the address information along with the compressed instructions in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading instructions from the memory. This information assigns the imaginary address of the decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the section in the memory and use it to assign the cache tag and therefore address of the cache block of the decompressed instruction in order to provide for faster lookups and shorter clock periods using simpler circuitry.

35. Regarding claim 8, Faraboschi has taught a processor as claimed in claim 7, wherein each said cache block has an associated cache tag in which is stored said imaginary block address assigned to the cache block with which the cache tag is associated (120 of Fig.2, also see Col.5 lines 48-50).

36. Regarding claim 9, Faraboschi has taught a processor as claimed in claim 5, but has not explicitly taught wherein said imaginary address information is contained in only a first one of said compressed sections to be loaded.

37. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see Fig.2 where code pointer segments have addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. However, in order to save space, only the location of the first entry of the section can be saved from which the other sections can be addressed using relative addressing. One of ordinary skill in the art would have recognized that by storing the address information along with the first one of the compressed sections in the memory, it would allow for faster lookups, savings in space as compared to storing the address information in all the sections, and enable easy setting of the cache tag when loading the cache with instructions from the memory. This information assigns the imaginary address of the decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the first one of the sections in the memory in order to allow for faster lookups and shorter clock periods using simpler circuitry.

38. Regarding claim 19, Faraboschi has taught a processor as claimed din claim 15, wherein the updating unit comprises:

a. A locating information register section (see 120, 150, and 152 of Fig.2) which stores said next-section-locating information associated with the most-recently-accessed cache block,

b. Said next-section-locating section being operable, in the event of such a cache miss, to employ the next-section-locating information stored in the location information register to located said position of said next compressed instruction (see Col.6 lines 11-24). Here, on a cache miss, the instruction address in the program counter is used to access the code pointer segment in the memory (see Col.6 lines 11-16). The code pointer is then used to locate the position in the memory of the next compressed section following the compressed section corresponding to that most-recently-accessed cache block (see Col.6 lines 16-24). The position located in the compressed memory is the next compressed section because compressed sections are arranged in consecutive memory locations (see Col.6 lines 20-24).

39. Faraboschi has not explicitly taught a copying section operable, when an instruction held in one of the cache blocks is fetched, to copy into the locating information register section said next-section-locating information stored in association with that block.

40. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the instruction memory (see lines in the code pointer segment having addresses in Fig.2) and that address information could easily be stored with the instruction to allow for faster

lookups in the memory just like a cache tag. One of ordinary skill in the art would have recognized that by storing the address information along with the first one of the compressed sections in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions from the memory (110 of Fig.2) by copying the information into the cache tag. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address information along with the first one of the sections in the memory and copy the address information into the cache tag in order to set it in a simple fashion, thus allowing for faster lookups and shorter clock periods, using simpler circuitry.

41. Regarding claim 21, Faraboschi has taught a processor as claimed in claim 20, but has not explicitly taught wherein the decompression section comprises a counter operable, during such a cache loading operation, to count the number of decompressed instructions that are not said pre-selected instructions.

42. However, "Official Notice" is taken that it is well known and expected in the art to sue a counter to count a number of instructions to be transferred from one location to another. One of ordinary skill in the art would have recognized that the cache refill unit uses the mask to find out how many decompressed instructions are to be transferred into the instruction cache (see Col.6 lines 27-39), and that a counter to count the number of decompressed instructions that are not NOP's could be advantageously used so as to transfer the right amount of instructions to the cache. Therefore, one of ordinary skill in the art would have found it obvious to have used a counter to count the number of decompressed instructions that are not said pre-selected instructions.

43. Regarding claim 27, Faraboschi has taught a method of compressing a program to be executed by a processor in which compressed-form instructions stored in a program memory (110 of Fig.3, also see Col.4 lines 45-46) are decompressed and cached in an instruction cache prior to being issued (see Col.5 lines 29-32), the method comprising:

a. Converting a sequence of original instructions of the program into a corresponding sequence of compressed-form instructions (see Col.1 line 44 – Col.2 line 22),

b. Assigning such original instructions imaginary addresses according to said sequence thereof, the assigned imaginary addresses being imaginary addresses at which the instructions are considered to exist when held in decompressed form in said instruction cache of the processor. Here, although not explicitly taught, it is inherent that during the compilation and storing of the program, the original instructions must be assigned addresses. As the decompressed form of the instructions are the same as the original form, the addresses will be the same.

44. Faraboschi has not explicitly taught wherein the method further comprises outputting a compressed program storable in said program memory and comprising the compressed-form instructions together with imaginary address information specifying said assigned imaginary address of at least one said original instruction so that, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache, the processor can allocate the assigned imaginary address to the decompressed instructions based on said imaginary address information.

45. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see lines in the code pointer segment having addresses in Fig.2) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the first one of the compressed sections in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions form the memory by copying the information into the cache tag. This information assigns the imaginary address of the decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the section in the memory and use it to assign the cache tag and therefore the address of the cache block of the decompressed instruction, thereby allowing for faster lookups and shorter clock periods using simpler circuitry.

46. Regarding claim 28, Faraboschi has taught a method as claimed in claim 27, wherein the assigned imaginary addresses are selected so that instructions likely to coexist in the instruction cache at execution time will not be mapped to the same cache block (see Fig.2). Because the cache is direct mapped, the addresses assigned will not be mapped to the same cache block.

47. Regarding claim 29, Faraboschi has taught a method as claimed in claim 27, wherein the compressed-form instructions are arranged to be stored in said program

memory in one or more compressed sections (see Fig.2), the compressed-form instructions belonging to each section occupying one cache block of the processor's instruction cache when decompressed (see Col.4 lines 48-67 and Col.5 lines 29-32), and at least one compressed section also containing imaginary address information relating to the instructions of that section (152 of Fig.2, imaginary address is stored in code pointer). Here, the program memory contains compressed-form instructions (W00-W56 of Fig.2) stored in a "compressed section" of the heap (see address 14000300 in heap of Fig.2), which are uncompressed and stored in a cache block of the cache (see WOO in row 040 of Fig.2).

48. Regarding claim 30, Faraboschi has taught a method as claimed in claim 29, but has not explicitly taught wherein said imaginary address information specifies the imaginary address at which a first one of the decompressed instructions corresponding to said at least one compressed section is to be considered to exists when the decompressed instructions are held in said instruction cache.

49. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see lines in the code pointer segment having addresses in Fig.2) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the compressed instructions in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions form the memory. This information assigns the imaginary address of the first one of the

decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the section in the memory, thereby allowing for faster lookups and shorter clock periods using simpler circuitry.

50. Regarding claim 31, Faraboschi has taught a method as claimed in claim 29, but has not explicitly taught wherein said imaginary address information is contained in only a first one of said compressed sections to be loaded.

51. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see Fig.2 where code pointer segments have addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. However, in order to save space, only the location of the first entry of the section can be saved from which the other sections can be addressed using relative addressing. One of ordinary skill in the art would have recognized that by storing the address information along with the first one of the compressed sections in the memory, it would allow for faster lookups, savings in space as compared to storing the address information in all the sections, and enable easy setting of the cache tag when loading the cache with instructions from the memory. This information assigns the imaginary address of the decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in

the art would have found it obvious to store the imaginary address information along with the first one of the sections in the memory in order to allow for faster lookups and shorter clock periods using simpler circuitry.

52. Regarding claim 32, Faraboschi has taught a method as claimed in claim 29, wherein each said compressed section contains imaginary address information (152 of Fig.2) relating to the instructions belonging to the section concerned. Here, the code pointer contains imaginary address information relating to the compressed instructions belonging to the section concerned.

53. Regarding claim 33, Faraboschi has taught a method as claimed in claim 29, wherein the or each said compressed section further contains a decompression key (150 of Fig.2) for use by the processor to carry out the decompression of the instructions belonging to said section (see Col.6 lines 29-34).

54. Regarding claim 34, Faraboschi has taught a method as claimed in claim 33, wherein said sequence of original instructions of the program comprises preselected instructions (NOP instructions indicated by the empty locations in cache of Fig.2) that are not stored explicitly in any said compressed section (see 140 of Fig.2), and the decompression key (150 of Fig.2) of the or each said compressed section identifies the positions at which said preselected instructions exist are to appear in a decompressed sequence of instructions corresponding to the section (see Col.5 lines 4-13). Here, the NOP instructions are not stored in the main (compressed) memory.

55. Regarding claim 35, Faraboschi has taught a method as claimed in claim 34, wherein said preselected instructions are "no operation" instructions (see Col.5 lines 4-13).

56. Regarding claim 38, Faraboschi has taught a processor, for executing instructions of a program stored in compressed form (see Col.4 lines 45-46, 58-67) in a program memory (see 110 of Fig.3), each said compressed-form instruction having an imaginary address at which the instruction is considered to exist when held in decompressed form within the processor (see Col.5 lines 19-21), and imaginary address information from which the imaginary addresses assigned to the compressed-form instructions is derivable (see Fig.2, as well as Col.4 lines 5-8 and Col.5 lines 29-32), said processor comprising:

a. A program counter (132 of Fig.2), which identifies a position in said program memory (see Col.5 lines 1-15 and Col.6 lines 15-27),

b. An instruction cache (100 of Fig.3), having a plurality of cache blocks (see Fig.2), each for storing one or more instructions of said program in decompressed form (see Col.4 lines 5-8 and Col.5 lines 29-35),

c. An imaginary address deriving unit operable to derive therefrom the imaginary address of at least a first one of the compressed-form instructions in said program (see Fig.2, as well as Col.4 lines 5-8 and Col.5 lines 29-32),

d. A cache loading unit (204/210/212 of Fig.3), comprising a decompression section (210/212 of Fig.3), operable to perform a cache loading operation in which one or more compressed-form instructions are read from said position in the program memory identified by the program counter and are decompressed and stored in one of said cache blocks of the instruction cache (see Col.5 lines 24-34 and Col.6 lines 11-53), which cache block is determined by the imaginary address of said one or more compressed-

form instructions being read from said position in the program memory (see Fig.2, as well as Col.4 line 48 – Col.5 line 21),

e. A cache pointer (200 of Fig.3), which identifies a position in said instruction cache of an instruction to be fetched for execution (see Col.5 lines 19-21, 55-57),

f. An instruction fetching unit (208/220 of Fig.3) which fetches an instruction to be executed from the position identified by the cache pointer (see Col.5 lines 36-40) and which, when a cache miss occurs because the instruction to be fetched is not present in the instruction cache, causes the cache loading unit to perform said cache loading operation (see Col.5 lines 40-50),

g. An updating unit (206 of Fig.3) which updates the program counter (132 of Fig.2) and cache pointer (200 of Fig.3) in response to the fetching of instructions so as to ensure that said position identified by said program counter is maintained consistently at the position in said program memory at which the instruction to be fetched form the instruction cache is stored in compressed form (see Col.5 lines 19-24 and Col.6 lines 11-53). Here, the program counter (132 of Fig.2), which is comprised of a mask (150 of Fig.2) and an offset (152 of Fig.2), are updated to point to the current position in the program memory when performing an instruction fetch upon a cache miss (see Col.6 lines 11-53). Thus, along with the updating of the cache pointer (see Col.5 lines 19-24), the position of the current instruction to be fetched from the instruction cache is maintained. Further,

> on a cache miss, the instruction address in the program counter (200 of Fig.3) is used to access the code pointer segment (130 of Fig.2) in program memory (see Col.6 lines 11-16). The code pointer (152 of Fig.2) is then used to locate the position in the program memory of the next compressed section following the compressed section corresponding to the most-recently-accessed cache block (see Col.6 lines 16-24). The position located in the compressed memory is considered the next compressed section because the compressed sections are arranged in consecutive memory locations (see Col.6 lines 20-24).

57. Faraboschi has not explicitly taught wherein the program memory stores imaginary address information or an imaginary address deriving unit operable to read the imaginary address information stored in the program memory.

58. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see Fig.2 where code pointer segments have addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art would have recognized that by storing the address information along with the compressed instructions in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions from the memory. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the compressed instruction, thus allowing for faster lookups and shorter clock periods using simpler circuitry.

59. Regarding claim 39, Faraboschi has taught a processor as claimed in claim 38, wherein:

a. The compressed-form instructions are stored in the program memory in one or more compressed sections (see Fig.2), the compressed-form instructions belonging to each section occupying one of said cache blocks when decompressed (see Col.4 lines 48-67 and Col.5 lines 29-32), and at least one section also contains imaginary address information relating to the instructions belonging to the section (see Fig.2). Here, the imaginary address information is stored in the code pointer (152 of Fig.2), and the program memory contains compressed-form instructions (W00-W56 of Fig.2) stored in a "compressed section" of the heap (see address 14000300 in heap of Fig.2), which are uncompressed and stored in a cache block of the cache (see WOO in row 040 of Fig.2).

b. Said cache loading unit is operable, in said cache loading operation, to decompress and load into one of said cache blocks one such compressed section stored at the position in the program memory identified by the program counter (see Col.5 lines 24-34 and Col.6 lines 11-53).

60. Regarding claim 40, Faraboschi has taught a processor as claimed in claim 39, but has not explicitly taught wherein said imaginary address information of said at least one section specifies the imaginary address at which a first one of the decompressed instructions corresponding to the compressed section is considered to exist when the decompressed instructions are held in one of the cache blocks.

61. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see Fig.2 where code pointer segments have addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art would have recognized that by storing the address information along with the compressed instructions in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading instructions from the memory. This information would specify the imaginary address of the first one of the decompressed instructions because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the section in memory in order to provide for faster lookups and shorter clock periods using simpler circuitry.

62. Regarding claim 41, Faraboschi has taught a processor as claimed in claim 39, wherein said imaginary address information is contained in only a first one of said compressed sections to be loaded (see Fig.2 and Col.4 lines 48-67 and Col.5 lines 29-32). Here, the imaginary address information is the mask (150 of Fig.2) and the code pointer (152 of Fig.2), and the mask for a section is only stored in association with the first compressed section (see "mask 132" of Fig.2 for example).

63. Regarding claim 42, Faraboschi has taught a processor as claimed in claim 39, wherein each said compressed section contains imaginary address information relating to the instructions belonging to the section concerned (see Fig.2 and Col.4 lines 48-67 and

Col.5 lines 29-32). Here, the imaginary address information is the mask (150 of Fig.2) and the code pointer (152 of Fig.2), and the mask for a section is only stored in association with the first compressed section (see "mask 132" of Fig.2 for example).

64. Regarding claim 43, Faraboschi has taught a computer-readable recording medium storing a compressed program, said compressed program being adapted to be stored in a program memory of a processor (see Col.4 lines 45-56) and comprising:

a. A sequence of compressed-form instructions derived from a corresponding sequence of original instructions (see Col.1 line 44 – Col.2 line 22), the compressed-form instructions being adapted to be decompressed by the processor (see Col.4 lines 45-56) and cached in an instruction cache thereof prior to issuance (see Col.5 lines 29-32),

b. Imaginary address information specifying an imaginary address assigned to at least one of said original instructions, being an imaginary address at which the original instruction is to be considered to exist when held in decompressed form in said instruction cache, whereby when the compressed-form instructions are decompressed (see Col.4 lines 45-56) and loaded by the processor into the instruction cache (see Col.5 lines 29-32). Here, although not explicitly taught, it is inherent that during the compilation and storing of the program, the original instructions must be assigned addresses. Thus, as the decompressed forms of the instructions are the same as the original form, the addresses will be the same.

65. Faraboschi has not explicitly taught wherein the processor can allocate the decompressed instructions such imaginary addresses based on said imaginary address information.

66. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see lines in the code pointer segment having addresses in Fig.2) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the first one of the compressed sections in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions form the memory by copying the information into the cache tag. This information assigns the imaginary address of the decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the section in the memory and use it to assign the cache tag and therefore the address of the cache block of the decompressed instruction, thereby allowing for faster lookups and shorter clock periods using simpler circuitry.

67. Regarding claim 44, Faraboschi has taught a computer-readable recording medium as claimed in claim 43, wherein the assigned imaginary addresses are selected so that instructions likely to coexist in the instruction cache at execution time will not be

mapped to the same cache block (see Fig.2). Because the cache is direct mapped, the addresses assigned will not be mapped to the same cache block.

68. Regarding claim 45, Faraboschi has taught a computer-readable recording medium as claimed in claim 43, wherein the compressed-form instructions are arranged to be stored in the said program memory in one or more compressed sections (see Fig.2), the compressed-form instructions belonging to each section occupying one cache block of the processor's instruction cache when decompressed (see Col.4 lines 48-67 and Col.5 lines 29-32), and at least one compressed section also containing imaginary address information relating to the instructions of that section (152 of Fig.2, imaginary address is stored in code pointer). Here, the program memory contains compressed-form instructions (W00-W56 of Fig.2) stored in a "compressed section" of the heap (see address 14000300 in heap of Fig.2), which are uncompressed and stored in a cache block of the cache (see WOO in row 040 of Fig.2).

69. Regarding claim 46, Faraboschi has taught a computer-readable recording medium as claimed in claim 45, but has not explicitly taught wherein said imaginary address information specifies the imaginary address at which a first one of the decompressed instructions corresponding to said one compressed section is to be considered to exist when the decompressed instructions are held in the same instruction cache.

70. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see lines in the code pointer segment having addresses in Fig.2) and that address information could easily be stored with the instruction to allow for faster

lookups in the memory just like a cache tag. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the compressed instructions in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions form the memory. This information assigns the imaginary address of the first one of the decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the section in the memory, thereby allowing for faster lookups and shorter clock periods using simpler circuitry.

71. Regarding claim 47, Faraboschi has taught a computer-readable recording medium as claimed in claim 45, but has not explicitly taught wherein said imaginary address information is contained in only a first one of the said compressed sections to be loaded.

72. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see Fig.2 where code pointer segments have addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. However, in order to save space, only the location of the first entry of the section can be saved from which the other sections can be addressed using relative addressing. One of ordinary skill in the art would have recognized that by storing the address information along with the first one of the compressed sections in the

memory, it would allow for faster lookups, savings in space as compared to storing the address information in all the sections, and enable easy setting of the cache tag when loading the cache with instructions from the memory. This information assigns the imaginary address of the decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the first one of the sections in the memory in order to allow for faster lookups and shorter clock periods using simpler circuitry.

73. Regarding claim 48, Faraboschi has taught a computer-readable recording medium as claimed in claim 45, wherein each said compressed section contains imaginary address information (152 of Fig.2) relating to the instructions belonging to the section concerned. Here, the code pointer contains imaginary address information relating to the compressed instructions belonging to the section concerned.

74. Regarding claim 49, Faraboschi has taught a computer-readable recording medium as claimed in claim 45, wherein the or each said compressed section further contains a decompression key (150 of Fig.2) for use by the processor to carry out the decompression of the instructions belonging to the said section (see Col.6 lines 29-34).

75. Regarding claim 50, Faraboschi has taught a computer-readable recording medium as claimed in claim 49, wherein said corresponding sequence of original instructions includes preselected instructions (NOP instructions indicated by the empty locations in cache of Fig.2) that are not stored explicitly in any said compressed section (see 140 of Fig.2), and the decompression key of the or each said compressed section

identifies the positions at which said preselected instructions exist are to appear in a decompressed sequence of instructions corresponding to the section (see Col.5 lines 4-13). Here, the NOP instructions are not stored in the main (compressed) memory.

76. Regarding claim 51, Faraboschi has taught a computer-readable recording medium as claimed in claim 50, wherein said preselected instructions are "no operation" instructions (see Col.5 lines 4-13).

77. Claims 22 and 24-26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Faraboschi et al., U.S. Patent No. 5,870,576 as applied to claim 1 above, and further in view of Guttag et al., U.S. Patent No. 5,509,129.

78. Regarding claim 22, Faraboschi has taught a processor as claimed in claim 1, but has not explicitly taught wherein the processor is operable to execute a hardware-controlled loop, wherein:

a. Said updating unit further comprises respective first and second loop control registers and operates, upon initiation of execution of such a hardware-controlled loop, to cause the program-counter value to be stored in said first loop control register and to cause the cache-pointer value to be stored in said second loop control register, and further operates, upon commencement of each iteration of the loop after said first iteration thereof, to reload said program counter with the value held in said first loop control register and to reload said cache pointer with the value held in said second loop control register.

79. However, Guttag discloses a program flow control unit in a long instruction word processor (see Guttag, Fig.31), where the program counter (see Guttag, 701 of Fig.31) is

connected to a loop control register (see Guttag, LS1 of Fig.31 and Col.162 line 13 – Col.163 line 50). During the initialization of the hardware controlled loop (see Guttag, Col.162 lines 41-44), the program counter value is stored in the loop control register (see Guttag, LS1 of Fig.31) and on commencement of each iteration of the loop (see Guttag, Col.163 lines 7-8, when loop count register equals zero), the program counter is reloaded with the loop control register value (see Guttag, Col.163 lines 7-14). Thus, Guttag detail a zero-overhead loop logic to control hardware branching (see Guttag, Col.162 lines 12-14). One of ordinary skill in the art would have recognized that the zero-overhead loop logic proposed by Guttag for a VLIW processor could be advantageously used in the Faraboschi VLIW processor to execute loops quickly. Therefore, it would have been obvious to one of ordinary skill in the art to have modified the update unit to have loop control registers for each of the program counter and cache pointer (as they both serve the same purpose of the program counter of Guttag) and to load the program counter and cache pointer values into their respective loop control register upon initiation of the loop and reload the program counter and cache pointer with the values in their respective loop control register on completion of an iteration of the loop as taught by Guttag, as the addition of the loop logic leads to zero-overhead execution of loops, thereby improving performance.

80. Regarding claim 24, Faraboschi has taught a processor as claimed in claim 1, but has not explicitly taught wherein the updating unit is operable, when an interrupt occurs during execution of a program, to cause the program-counter value and cache-pointer value to be saved pending handling of the interrupt, and, when the execution of the

program is resumed, to cause the saved values to be restored in the program counter and cache pointer.

81. However, Guttag discloses an interrupt handling mechanism (see Guttag, 706, 707, 770, 702, 703, 701 and 704 of Fig.31) wherein when an enabled interrupt occurs and the interrupt service routine branches to another location, the program counter (see Guttag, 701 of Fig.31) value is stored in an instruction pointer-return from subroutine register (see Guttag, 704 of Fig.31) in order to be able to use that register value to return form the interrupt service routine by restoring the program counter contents (see Guttag, Col.102 lines 35-56 and Col.110 lines 28-30, 50-56). One of ordinary skill in the art would have easily recognized that the interrupt handling technique propose by Guttag for a VLIW processor could be advantageously used in the Faraboschi VLIW processor to enable the handling of interrupt requests and allow for I/O interrupt handling, exception handling, and other well known useful services in the art. Therefore, one of ordinary skill in the art would have found it obvious to have modified the update unit to have an interrupt handling mechanism to save the values of the program counter and cache pointer (as they both serve the save purpose of the program counter of Guttag) in an IPRS register on an interrupt and further using those saved values to restore the contents of the program counter and cache pointer to resume execution of the program, as the addition of the interrupt handling mechanism would enable the handling of interrupt requests and increase the functionality of the VLIW processor.

82. Regarding claim 25, Faraboschi has taught a processor as claimed in claim 1, but has not explicitly taught wherein the updating unit is operable, when an interrupt occurs during execution of a program, to cause said next-section locating information associated

with the most-recently-accessed cache block to be saved pending handling of the interrupt, and, when execution of the program is resumed, to cause the saved next-section locating information to be restored.

83. However, Guttag discloses an interrupt handling mechanism (see Guttag, 706, 707, 770, 702, 703, 701 and 704 of Fig.31) wherein when an enabled interrupt occurs and the interrupt service routine branches to another location, the program counter value (see Guttag, 701 of Fig.31), which is the next-section locating information, is stored in an instruction pointer-return from subroutine register (see Guttag, 704 of Fig.31) in order to be able to use that register value to return from the interrupt service routine by restoring the program counter contents (see Guttag, Col.102 lines 35-56 and Col.110 lines 28-30, 50-56). One of ordinary skill in the art would have easily recognized that the interrupt handling technique propose by Guttag for a VLIW processor could be advantageously used in the Faraboschi VLIW processor to enable the handling of interrupt requests and allow for I/O interrupt handling, exception handling, and other well known useful services in the art. Therefore, one of ordinary skill in the art would have found it obvious to modify the update unit to have an interrupt handling mechanisms to save the values of the next-section locating information, i.e. the program counter (see Faraboschi, 200 of Fig.2) and cache pointer (see Faraboschi, 152 of Fig.2), as they both serve the same purpose of the program counter of Guttag, on an interrupt and further using those saved values to restore the contents of the program counter and cache pointer to resume execution of the program, as the addition of the interrupt handling mechanism would enable the handling of interrupt requests and increase the functionality of the VLIW processor.

84. Regarding claim 26, Faraboschi has taught the processor of claim 20, but has not explicitly taught wherein the updating unit is operable, when an interrupt occurs during execution of a program, to cause the values held in said loop control registers to be saved pending handling of the interrupt, and, when execution of the program is resumed, to cause the saved values to be restored in the loop control registers.

85. However, Guttag discloses an interrupt handling mechanism (see Guttag, 706, 707, 770, 702, 703, 701 and 704 of Fig.31) wherein when an enabled interrupt occurs and the interrupt service routine branches to another location, the program counter value (see Guttag, 701 of Fig.31), which is the value stored in the loop control registers (see Guttag, 721-723 of Fig.31), is stored in an instruction pointer-return from subroutine register (see Guttag, 704 of Fig.31) in order to be able to use that register value to return form the interrupt service routine by restoring the program counter contents (see Guttag, Col.102 lines 35-56 and Col.110 lines 28-30, 50-56). One of ordinary skill in the art would have easily recognized that the interrupt handling technique proposed by Guttag for a VLIW processor could be advantageously used in the Faraboschi VLIW processor to enable the handling of interrupt requests and allow for I/O interrupt handling, exception handling, and other well known useful services in the art. Therefore, one of ordinary skill in the art would have found it obvious to modify the update unit to have an interrupt handling mechanism to save the values of the loop control registers which store the program counter (see Faraboschi, 200 of Fig.3) and cache pointer (see Faraboschi, 152 of Fig.2) values on an interrupt and further using those saved values to restore the contents of the loop control registers to resume execution of the program, as the addition of the interrupt

handling mechanism would enable the handling of interrupt requests and increase the functionality of the VLIW processor.

86. Claim 36 is rejected under 35 U.S.C. 103(a) as being unpatentable over Faraboschi et al., U.S. Patent No. 5,870,576, in view of Tannenbaum, *Structured Computer Organization*.

87. Regarding claim 36, Faraboschi has taught a computer-readable recording medium storing a computer program which carries out a method of compressing a processor program to be executed by a process, the processor being operable to decompress compressed-form instructions stored in a program memory (110 of Fig.3, also see Col.4 lines 45-46) and to cache the decompressed instructions in an instruction cache prior to issuing them (see Col.5 lines 29-32), the computer program comprising:

a. A converting portion which converts a sequence of original instructions of the processor program into a corresponding sequence of such compressed-form instructions (see Col.1 line 44 – Col.2 line 22),

b. An assigning portion which assigns such original instructions imaginary addresses according to said sequence thereof, the assigned imaginary addresses being imaginary address at which the instructions are to be considered to exist when held in decompressed form in said instruction cache of the processor. Here, although not explicitly taught, it is inherent that during the compilation and storing of the program, the original instructions must be assigned addresses. As the decompressed forms of the instructions are the same as the original form, the addresses will be the same.

88. Faraboschi has not explicitly taught wherein the method further comprises an outputting portion which outputs a compressed program storable in said program memory and comprising the compressed-form instructions together with imaginary address information specifying said assigned imaginary address of at least one said original instruction so that, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache, the processor can allocate the assigned imaginary address to the decompressed instructions based on said imaginary address information. Also, Faraboschi has not taught wherein the instructions on a computer-readable medium carry out the method of the invention as claimed.

89. However, "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of an instruction to be accessed in the instruction memory (see lines in the code pointer segment having addresses in Fig.2) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the first one of the compressed sections in the memory (110 of Fig.2), it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions form the memory by copying the information into the cache tag. This information assigns the imaginary address of the decompressed instruction because the program counter (200 of Fig.3) points to the cache block where the instruction is to be stored and the location in memory where the instruction information is stored. Therefore, one of ordinary skill in the art would have found it obvious to store the imaginary address information along with the section in the memory and use it to assign the cache tag and

therefore the address of the cache block of the decompressed instruction, thereby allowing for faster lookups and shorter clock periods using simpler circuitry.

90. Furthermore, Tannenbaum has taught that any instruction executed by hardware can also be simulated in software (see Tannenbaum, p.11, para. 4, lines 1-2). He also has taught that hardware is generally immutable (see first para. after sec. 1.4 header), while software allows for more rapid change (see Tannenbaum, p.11, para. 4, lines 1-2). One of ordinary skill in the art at the time of the invention would have been motivated to convert Faraboschi to software, i.e. instructions on a machine-readable medium because Tannenbaum has taught that hardware is generally immutable while software allows for more rapid changes. Therefore, one of ordinary skill in the art would have found it obvious to modify Faraboschi to be instructions recorded on a machine readable medium in the manner suggested by Tannenbaum in order to allow for ease of correction of mistakes and/or an ease of addition of new functionality.

Response to Arguments

91. Applicant's arguments filed 6/29/2004 have been fully considered but they are not persuasive.

92. In the first paragraph on p.30, the Applicant argues, in essence:

> *"The present invention is concerned with a processor that is capable of performing "on-the-fly" decompression. When a cache miss occurs, the decompression of the compressed-form instructions and the loading of the decompressed instructions into the instruction cache are time-critical operations for the processor. ... In Faraboschi, the next-section locating information is the ptr 152, which is held outside the processor in the main memory 110. As a result, there is a*

delay in accessing this information and consequently a delay in commencing the decompression and loading processes."

93. In response to applicant's argument that the references fail to show certain features of applicant's invention, it is noted that the features upon which applicant relies (i.e., "on-the-fly decompression", "time-critical operations", and the "next-section locating information being held within the processor) are not recited in the rejected claim(s). Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993).

94. In the last two paragraphs of p.30 and the first two paragraphs on p.31, the Applicant argues with respect to claim 1, in essence:

"*The decompression processes of the present invention and Faraboschi are fundamentally different. In Faraboschi, the aim is to have the program counter incremented by a fixed amount each instruction cycle, so that it simply moves through sequential locations in the code pointer segment 130 of the main memory 110. This provides the advantage that control of the program counter is simple and quick. However, the penalty for this is that there must be and address mapping operation from the location in the code pointer segment 130 to the relevant location in the code heap segment 140, and this causes some delay. ... In the present invention, the program counter points directly to the position of the next set of compressed-form instructions in the program memory. This means that the program counter cannot simply be incremented by a fixed amount each instruction cycle. Thus, there is a penalty in terms of complexity of program counter manipulation. However, the advantage of doing this is that the program counter is maintained consistently at the position the program memory where the next instruction is stored in compressed form. ... In view of the fundamentally different approaches in Faraboschi and the present invention, a skilled worker would not seek to modify Faraboschi to arrive at a processor*

in accordance with amended claim 1, since to do so would required him to discard the entire approach taught by Faraboschi."

95. However, no modification to Faraboschi is required to read upon claim 1 (see above paragraph 11). Faraboschi has taught all of the limitations of amended claim 1, and thus no "discarding" of the approach taught by Faraboschi is required.

Conclusion

96. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

97. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. Applicant is reminded that in amending in response to a rejection of claims, the patentable novelty must be clearly shown in view of the state of the art disclosed by the references cited and the objections made. Applicant must also show how the amendments avoid such references and objections. See 37 CFR § 1.111(c).

98. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Barry J. O'Brien whose telephone number is (703) 305-5864. After October 12th, 2004, the examiner can be reached at (571) 272-4171. The examiner can normally be reached on Mon.-Fri. 6:30am-4:00pm, with the exception of first Friday of every bi-week.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached at (703) 305-9712, or at (571) 272-4162 on or after October 12th, 2004. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

99. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Barry J. O'Brien
Examiner
Art Unit 2183

BJO
9/8/2004

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100